Growing Better Graphs With Latent-Variable Probabilistic Graph Grammars

Xinyi Wang
Carnegie Mellon University
Pittsburgh, PA
xinyiw1@cs.cmu.edu

Tim Weninger
University of Notre Dame
Notre Dame, IN
tweninge@nd.edu

Salvador Aguinaga
University of Notre Dame
Notre Dame, IN
saguinag@nd.edu

David Chiang
University of Notre Dame
Notre Dame, IN
dchiang@nd.edu

ABSTRACT

Recent work in graph models has found that probabilistic hyperedge replacement grammars (HRGs) can be extracted from graphs and used to generate new random graphs with graph properties and substructures close to the original. In this paper, we show how to add latent variables to the model, trained using Expectation-Maximization, to generate still better graphs, that is, ones that generalize better to the test data. We evaluate the new method by separating training and test graphs, building the model on the former and measuring the likelihood of the latter, as a more stringent test of how well the model can generalize to new graphs. On this metric, we find that our latent-variable HRGs consistently outperform several existing graph models and provide interesting insights into the building blocks of real world networks.

1 INTRODUCTION

Detecting, interpreting and comparing structures and properties of network data about social interactions and complex physical phenomena is critically important to a variety of problems. However, this is a difficult task because comparisons between two or more networks can involve checking for graph or subgraph isomorphism, for which no tractable solution is known. Instead, various network properties (*e.g.*, degree distribution, centrality distributions) have been used to describe and compare networks.

Another approach is to consider a network's global structure as a by-product of a graph's local substructures [22]. More sophisticated graph statistics are based on counting the number of small motifs [16] or graphlets [4] present in the graph and comparing their distributions [23]. Unfortunately, graphlet counting presupposes that all possible graphlets be enumerated ahead of time. Because the number of unique graphlets increases exponentially with the number of nodes in the graphlet, previous work has been limited to graphlets of at most five nodes.

An alternative to developing sophisticated graph statistics is to learn graph generation models that encode properties of the graph in various ways. Graph generators like the Exponential Random Graph Model (ERGM) [20], the Chung-Lu Edge Configuration Model (CL) [8], the Stochastic Kronecker Graph (SKG) [12], and the Block Two-Level Erdős-Rényi Model (BTER) [21] can be fitted to real-world graphs.

Recent work has found that many social and information networks have a more-or-less tree-like structure, which implies that detailed topological properties can be identified and extracted from a graph's *tree decomposition* [1]. Based on these findings, Aguinaga *et al.* described a method to turn a graph's tree decomposition into a *Hyperedge Replacement Grammar* (HRG) [2]. The HRG model can then generate new graphs with properties similar to the original.

One limitation of the HRG model is that the instructions for reassembling the building blocks, *i.e.*, the graph grammar, encode only enough information to ensure that the result is well-formed. HRG production rules are extracted directly from the tree decomposition; some rules from the top of the tree, some from the middle of the tree, and some from the bottom of the tree. Then, to generate an new graph that is similar to the original graph, we would expect that rules from the top of the tree decomposition are applied first, rules from the middle next, and rules from the leaves of the tree are applied last. However, when generating a new graph the HRG model applies rules probabilistically; where a rule's probability relative to its frequency in the grammar. However, when generating a new graph, the rules in HRG models have no context on when they should fire. HRG models are in need of a mechanism that corrects for this problem by providing context to the rules.
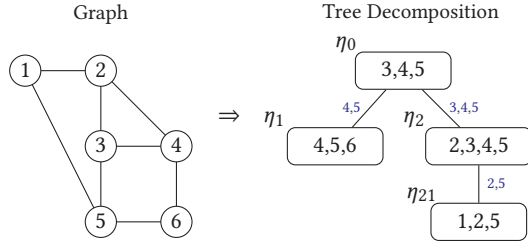
In the present work, we make three contributions:

(1) We improve the HRG model by encoding context in latent variables.
(2) We propose a methodology for evaluating our model that enforces a strict separation between training and test data, in order to guard against overfitting.

Figure 1: An example graph and its tree decomposition. The width of this tree decomposition is 3, *i.e.*, the size of the largest bag minus 1. The sepset between each bag and its parent is labeled in blue. Bags are labeled (η_0, etc.) for illustration purposes only.

(3) We test our model on 6 train/test pairs of graphs and find that it discovers a better model, that is, one that generalizes better to the test data, than the original HRG model as well as Kronecker and Chung-Lu models.

2 BACKGROUND

Before we introduce our model, we first provide an overview and examples of the HRG model.

2.1 Hyperedge Replacement Grammars

Like a context free string grammar (CFG), an HRG has a set of production rules $A \to R$, where A is called the left-hand side (LHS) and R is called the right-hand side (RHS). In an HRG, a rule's RHS is a graph (or hypergraph) with zero or more *external* nodes. Applying the rule replaces a hyperedge labeled A with the graph R; the nodes formerly joined by the hyperedge are merged with the external nodes of R. The HRG generates a graph by starting with the start nonterminal, S, and applying rules until no more nonterminal-labeled hyperedges remain.

2.2 Tree Decomposition

Given a graph $H = (V, E)$, a *tree decomposition* is a tree whose nodes, called *bags*, are labeled with subsets of V, in such a way that the following properties are satisfied:

- For each node $v \in V$, there is a bag η that contains v.
- For each edge $(u, v) \in E$, there is a bag η that contains u and v.
- If bags η and η' contain v, then all the bags on the path from η to η' also contain v.

If η' is the parent of η, define $\bar{\eta} = \eta' \cap \eta$ (also known as the *sepset* between η' and η). If η is the root, then $\bar{\eta} = \emptyset$.

All graphs can be decomposed (though not uniquely) into a tree decomposition, as shown in Fig. 1. In simple terms, a tree decomposition of a graph organizes its nodes into overlapping *bags* that form a tree. The *width* of the tree decomposition, which is related to the size of the largest bag, measures how tree-like the graph is. Finding optimal tree decompositions is NP-hard, but there is significant interest in finding fast approximations because many computationally difficult problems can be solved efficiently when the data is constrained to be a tree-like structure.
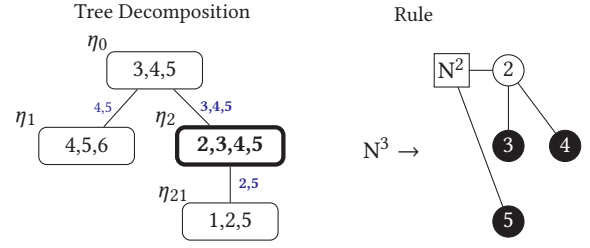
Figure 2: Extraction of an HRG production rule from η_2 containing graph vertices {2,3,4,5}. The LHS of the production rule corresponds to the sepset of the bag and its parent. The RHS of the production rule contains nodes from the bag, terminal edges induced from the original graph, and nonterminal edges from the sepset between the bag and its children.

2.3 Grammar Extraction

Aguinaga et al. [2] extract HRG rules from a graph, guided by a tree decomposition of the graph. For example, Figure 2 illustrates how one HRG rule is extracted from a tree decomposition.

If we assume that the tree decomposition is rooted, then every bag η of the tree decomposition corresponds to an edge-induced subgraph, which we write G_η, defined as follows: For each edge $(u, v) \in E$, if every bag η' containing u, v is either equal to η or a descendant of η, then $(u, v) \in H_\eta$. For example, in Figure 1, the bag $\eta_2 = \{2, 3, 4, 5\}$ corresponds to the subgraph H_{η_2} whose edges are 1–2, 1–5, 2–3, 2–4, and 3–5.

If $H = (V, E)$ is a graph and $H' = (V', E')$ is an edge-induced subgraph of H, we define an *external* node of H' to be any node of H' that has a neighbor not in H'. Then, define the operation of *replacing* H' with a hyperedge to be:

- Remove all edges in E'.
- Remove all nodes in V' except for the external nodes.
- Add a hyperedge joining the external nodes.

Every bag η also induces a HRG rule $N^{|\bar{\eta}|} \to R$, where R is constructed as follows.

- Make a copy of H_η.
- Label the nodes in $\bar{\eta}$ as external nodes.
- For each child η_i of η, replace H_{η_i} with a hyperedge labeled $N^{|\bar{\eta}_i|}$.

For example, in Figure 2, the bag η_2 induces the rule shown at right. The LHS is N^3 because the sepset between η_2 and its parent has three nodes (3, 4, 5); in the RHS, these three nodes are marked as external. The node numbers are for illustration purposes only; they are not actually stored with the production rules. The RHS has two terminal edges (2–3, 2–4) from the original graph and one nonterminal edge (2–5) corresponding to the sepset between η_2 and its one child.

After an HRG is extracted from the tree decomposition, its production rules are gathered into a set, merging identical rules and assigning to each unique rule $(A \to R)$ a probability $P(A \to R) = P(R \mid A)$ proportional to how many times it was encountered during extraction. This grammar can then be used to randomly generate new graphs, or compute the probability of another graph.
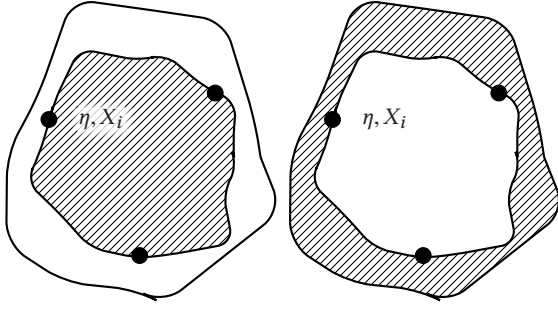
Figure 3: Inside and outside probabilities. Here, a hyperedge labeled X_i with three external nodes has generated a subgraph η. The inside probability (left) of η, X_i is the probability of all the subderivations that generate subgraph η from X_i. The outside probability (right) is the probability of all the partial derivations that generate the complement of subgraph η, with a hyperedge labeled X_i in place of the subgraph.

Figure 4: Computation of inside and outside probabilities. Here, a hyperedge labeled X_i has been rewritten with a rule rhs with two hyperedges labeled Y_j and Z_k. At left, the inside probability of η, X_i is incremented by the product of the rule and the inside probabilities of η_1, Y_j and η_2, Z_k. At right, the outside probability of η_1, Y_j is incremented by the product of the outside probability of η, X_i, the rule, and the inside probability of η_2, Z_k.

3 LATENT VARIABLE PROBABILISTIC HYPEREDGE REPLACEMENT GRAMMARS

Here, we improve upon the HRG model by encoding more context into the model via latent variables, in a process that is analogous to how a first-order Markov chain can simulate a higher-order Markov chain by expanding the state space.

In this section, we adopt a notational shortcut. In an HRG production $A \rightarrow R$, the RHS R is a hypergraph fragment containing zero or more hyperedges with nonterminal labels $Y_1, \ldots, Y_r$. We suppress the graph structure of R and write the rule simply as $X \rightarrow Y_1 \cdots Y_r$.
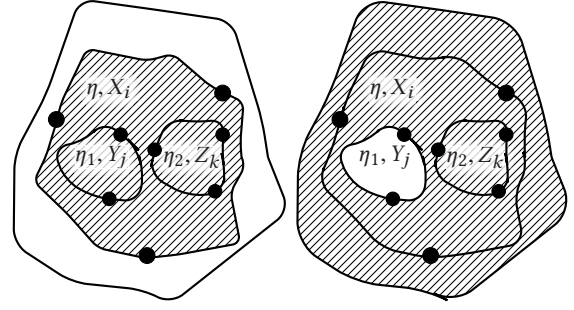
3.1 Nonterminal Splitting

Following previous work on probabilistic CFGs [15, 17], we increase the context-sensitivity of the grammar by splitting each nonterminal symbol X in the grammar into n different *subsymbols*, $X_i, \ldots, X_n$, which could potentially represent different contexts that the rule is used in. Thus, each rule in the original grammar is replaced with several *subrules* that use all possible combinations of the subsymbols of its nonterminal symbols.

For example, if $n = 2$, the rule $N^2 \rightarrow \epsilon$ would be split into $N_1^2 \rightarrow \epsilon$ and $N_2^2 \rightarrow \epsilon$.

In general, a rule with r nonterminal symbols on its right-hand side is split into n^{r+1} subrules.

3.2 Learning

After obtaining an n-split grammar from the training graphs, we want to learn probabilities for the subrules that maximize the likelihood of the training graphs and their tree decompositions. Here we use Expectation-Maximization (EM) [9]. In the E (Expectation) step, we use the Inside-Outside algorithm [11] to compute the expected count of each subrule given the training data, and in the M (Maximization) step, we update the subrule probabilities by normalizing their expected counts.

These expected counts can be computed efficiently using dynamic programming. Given a tree decomposition T, consider a bag η and its corresponding subgraph H_η. The grammar extraction method of Background Section 2.3 assigns H_η a nonterminal symbol, which we write X. Let X_i be a subsymbol of X. The *inside* probability of H_η with label X_i, written as $P_{in}(\eta, X_i)$, is the total probability of all derivations starting from X_i and ending in H_η. The *outside* probability of H_η with label X_i, written as $P_{out}(\eta, X_i)$, is the total probability of all derivations starting from S and ending in H with H_η replaced with a hyperedge labeled X_i. See Figure 3.

The inside probabilities can be calculated recursively, from smaller subgraphs to larger subgraphs. We assume that bag η has at most two children, which follows if T is in Chomsky Normal Form [7]. If η has two children, let η_1 and η_2 be the children, let Y and Z be the labels of H_{η_1} and H_{η_2}, and let Y_j be and Z_k be subsymbols of Y and Z. Then the inside probability of H_η with subsymbol X_i is defined by:

$$P_{in}(\eta, X_i) = \sum_{j,k} P(X_i \rightarrow Y_j Z_k) \, P_{in}(\eta_1, Y_j) \, P_{in}(\eta_2, Z_k)$$

and similarly if η has only one child:

$$P_{in}(\eta, X_i) = \sum_{j} P(X_i \rightarrow Y_j) \, P_{in}(\eta_1, Y_j)$$

or no children:

$$P_{in}(\eta, X_i) = P(X_i \rightarrow \epsilon).$$

The outside probabilities are calculated top-down. If a bag η has two children, then the outside probabilities of its children are defined by:

$$P_{out}(\eta_1, Y_j) = \sum_{i,k} P(X_i \rightarrow Y_j Z_k) \, P_{out}(\eta, X_i) \, P_{in}(\eta_2, Z_k)$$

$$P_{out}(\eta_2, Z_k) = \sum_{i,j} P(X_i \rightarrow Y_j Z_k) \, P_{out}(\eta, X_i) \, P_{in}(\eta_1, Y_k).$$

See Figure 4 for an illustration of this computation. Similarly, if η has only one child:

$$P_{out}(\eta_1, Y_j) = \sum_i P(X_i \to Y_j) \, P_{out}(\eta, X_i).$$

In the Expectation step, we compute the posterior probability of each subrule at each bag of each training tree decomposition T:

$$P(\eta, X_i \to Y_j Z_k \mid T) = \frac{1}{P(T)} P_{out}(\eta, X_i) P(X_i \to Y_j Z_k) \cdot$$
$$P_{in}(\eta_1, Y_j) P_{in}(\eta_2, Z_k)$$

$$P(\eta, X_i \to Y_j \mid T) = \frac{1}{P(T)} P_{out}(\eta, X_i) P(X_i \to Y_j) P_{in}(\eta_1, Y_j)$$

$$P(\eta, X_i \to \epsilon \mid T) = \frac{1}{P(T)} P_{out}(\eta, X_i) P(X_i \to \epsilon)$$

where $P(T) = P_{in}(\eta_0, S)$ and η_0 is the root bag of T. The expected count of each subrule is calculated by summing over the posterior probability of the rule over all nodes of all training trees:

$$E[c(X_i \to \alpha)] = \sum_{\text{trees } T} \sum_{\eta \in T} P(\eta, X_i \to \alpha \mid T)$$

where α is any right-hand side.

In the Maximization step, we use the expected counts calculated above to update the rule probabilities:

$$P(X_i \to \alpha) := \frac{E[c(X_i \to \alpha)]}{\sum_{\alpha'} E[c(X_i \to \alpha')]}.$$

These probabilities are then used to repeat the E step. The method is guaranteed to converge to a local maximum of the likelihood function, but not necessarily to a global maximum.

4 EVALUATION

Current research in graph modelling and graph generation evaluate their results by comparing the generated graphs with the original graph by aggregate properties like degree distribution, clustering coefficients, or diameter [2, 3, 5, 12, 13, 21]. There are two potential problems with such metrics. First, these metrics do not test how well the model generalizes to model other graphs that represent similar data. Second, they are heuristics from which a generated graph's "goodness" is difficult to define or standardize. We discuss and address both of these problems in this section.

4.1 Train/Test Data

Comparing generated graphs with the original graph cannot test how well the model generalizes to other graphs that represent similar data or different versions of the same network phenomena. To see why, consider the extreme case, in which a model simply memorizes the entire original graph. Then, the generated graphs are all identical to the original graph and therefore score perfectly according to these metrics. This is akin to overfitting a machine learning classifier on training data and then testing it on the same data, which would not reveal whether the model is able to generalize to unseen instances.

In standard data mining and machine learning tasks, the overfitting problem is typically addressed through cross-validation or by evaluating on heldout test data sets. In the present work, we adapt the idea of using heldout test data to evaluate graph grammars. In experiments on synthetic graphs, this means that we generate

two random graphs using the same model and parameters; we designate one as the training graph and the other as the test graph. In experiments on real world graphs, we identify two graphs that represent the same phenomenon, *e.g.*, citations or collaborations, and we mark one as the training graph and one as the test graph.

In reality, we might not be able to find test graphs that have similar properties as the training graph. Fortunately, cross-validation can also be adapted to cases where no test graph is available by using disjoint subgraph samples from a single graph.

4.2 Likelihood

In addition to the possibility of overfitting, high-level aggregations of graph properties may not always be good comparators of two or more graphs. Indeed, examples abound in related literature showing how vastly different graphs can share similar aggregate statistics [23]. We propose, as an additional metric, to evaluate models by using them to measure the likelihood of a test graph or graphs. Intuitively, this measures how well a model extracted from the training graph generalizes to a test graph. If the model simply memorizes the entire training graph, then it will have zero likelihood (the worst possible) on the test graph. If the model is better able to generalize to new graphs, then it will have higher likelihood on the test graph.

Unfortunately, it is not always computationally feasible to compute the likelihood of graphs under previous models. But with HRGs, it can be computed in linear time given a tree decomposition. (It would also be possible, but slower, to sum the probabilities of *all* possible tree decompositions [6].) The likelihood on a test graph is simply $P_{in}(\eta_0, S)$, where η_0 is the root of the tree decomposition. Note that the model probabilities are estimated from the training graphs, even when computing likelihood on test graphs. As this number is usually very small, it's common to take logs and deal with log-likelihoods.

4.3 Smoothing

A problem arises, however, if the test graph uses a rule that does not exist in the grammar extracted from the training graph. Then the inside probability will be zero (or a log-probability of $-\infty$). This is because an HRG missing any necessary rules to construct the test graph cannot generate the test graph exactly, and therefore results in a zero probability.

In this case, we would still like to perform meaningful comparisons between models, if possible. So we apply smoothing as follows. To test an HRG H on a test graph, we first extract an HRG, H', from the test graph using the latent-variable HRG method. Define an *unknown rule* to be a rule in H' but not in H. Then for each unknown rule, we add the rule to H with a probability of ϵ. We can then compute the log likelihood on the test graph under the augmented grammar $H \cup H'$. The final test log likelihood is calculated as

$$L = L_{H \cup H'} - c(H' \setminus H) \cdot \log \epsilon$$

where $L_{H \cup H'}$ is the log likelihood of the test graph under the augmented grammar, $c(H' \setminus H)$ is the number of times that unknown rules are used in the test graph, and ϵ is the probability of each unknown rule. Note that as long as ϵ is much smaller than the

Table 1: Datasets used in experiments

	Type	Name	Nodes	Edges
Synth	Barabasi-Albert	train-ba	30,000	59,996
		test-ba	30,000	59,996
	Watts-Strogatz	train-ws	30,000	60,000
		test-ws	30,000	60,000
Real	Citation	train-cit-HepTh	27,770	352,807
		test-cit-HepPh	34,546	421,578
	Internet	train-as-topo	34,761	171,403
		test-as-733	6,474	13,895
	Purchase	train-amazon0312	400,727	3,200,440
		test-amazon0302	262,111	1,234,877
	Wikipedia	train-wiki-vote	7,115	103,689
		test-wiki-talk	2,394,385	5,021,410

probability of any known rule, its value is irrelevant because L does not depend on it.

Ideally, we would like the number of unknown rules to be zero. In our experiments, we find that increasing the number of training graphs and/or decreasing the size of the training graphs can bring this number to zero or close to zero. Note that if two HRGs have differing sets of unknown rules, then it is *not* meaningful to compare their log-likelihood on test graphs. But if two HRGs have identical sets of unknown rules, then their log-likelihoods can be meaningfully compared. We will exploit this fact when evaluating models with latent variables in the next section.

5 EXPERIMENTS

In this section we test the ability of the latent-variable HRG (laHRG) to generate graphs using the train-test framework described above. We vary n, the number of subsymbols that each nonterminal symbol is split into, from 1 to 4. Note that the 1-split laHRG model is identical to the original HRG method. By varying the number of splits, we will be able to find the value that optimizes the test likelihood. We have provided all of the source code and data analysis scripts at https://github.com/cindyxinyiwang/laHRG.

5.1 Setup

Given a training graph, we extract and train a latent-variable HRG from the graph. Then we evaluate the goodness of the grammar by calculating the log likelihood that the test graph could be generated from the grammar.

For evaluation, we need to make sure that the test graph is disjoint from the training graph to guard against overfitting. Here we introduce two techniques to achieve that: 1. we partition a single graph into two disjoint parts so that they do not have overlapping vertices. Then we train laHRG from one part of the graph, and calculate the log likelihood of the other disjoint part; 2. we choose 2 graphs of the same type, and use one for training and the other for evaluation.

We evaluate laHRG with log likelihood metric for both evaluation methods mentioned above on 6 types of graphs: 2 synthetic graphs (generated from random graph generators), and 4 real world

graphs. For the first evaluation method, we use 1 graph, each from 6 different types of graphs, and partition the graph for training and testing purpose. For the second evaluation method, we do not partition the training graph, but choose 1 additional graph from each type of graphs for testing. The graphs were obtained from the SNAP[1] and KONECT[2] graph repositories and are listed in Table 1.

Many of these graphs are too large for a tree decomposition to be calculated. Instead, we randomly sampled a set of fixed-size subgraphs from the training graph and a set of fixed-size subgraphs from the test graph. Besides the concern from the calculation of large tree decompositions, sampling multiple graphs is also important for the extraction of a broad set of rules. Recall that if a single rule required to generate the test graph is not found within the training graph, then the likelihood will be 0. Therefore, large test graphs would require many more training graphs in order to reduce (or hopefully eliminate) the need for smoothing.

In all experiments, we extract 500 samples of size-25 subgraphs from the training graph. We extract an HRG from each size-25 subgraph, perform nonterminal splitting and EM training. The 500 HRGs are then combined and their weights are normalized to create the final laHRG model. We also take 4 samples of size-25 subgraphs from the test graph, calculate the log-likelihood of each under the laHRG model, and report the mean log-likelihood and confidence interval.

We chose these parameters empirically such that there is no need for smoothing. If we were to increase the subgraph size for the test graphs, then we would also need to increase the number of training graph samples or rely on smoothing to ensure non-zero likelihood.

To compute tree decompositions, we used a reimplementation of the QuickBB algorithm [10], with only the "simplicial" and "almost-simplicial" heuristics.

5.2 Log-Likelihood Results

This section explains the performance of laHRG in terms of log-likelihood metric on test graph for two different train-test split methods mentioned in the previous section. We mainly analyze the results for the second method: train on one graph and test on another graph of the same type. The first method has similar results, and we include them here to show that our evaluation method also works for graphs that are difficult to find different test graphs of the same type.

5.2.1 Validate on Different Graph of Same Type. We first show the log-likelihood results on synthetic datasets. The two random graph models, the Barabasi-Albert graph and Watts-Strogatz graph, generate very different graph types. The four panels in Fig. 5 show the log-likelihood results of four combinations of training graphs and test graphs. Higher is better.

As a sanity check, we also trained an laHRG model on a Barabasi-Albert graph and tested it against a Watts-Strogatz graph and vice versa. We expect to see much lower log-liklihood scores because the laHRG trained on one type of graph should be different than another type of graph. The top-right and bottom-left panels in Fig. 5

---

[1] https://snap.stanford.edu/data
[2] http://konect.uni-koblenz.de
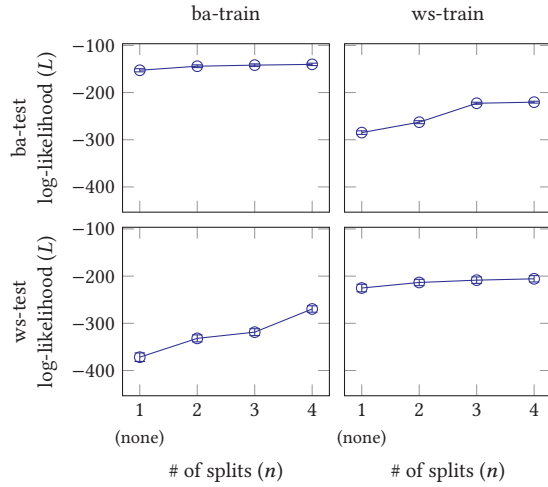
Figure 5: On synthetic graphs, splitting nonterminal symbols ($n \geq 2$) always improves log-likelihood on the test graph, as compared to no splitting ($n = 1$). We did not observe overfitting up to $n = 4$. Left/right column: train on Barabasi-Albert (ba) or Watts-Strogatz (ws). Top/bottom row: test on ba/ws.
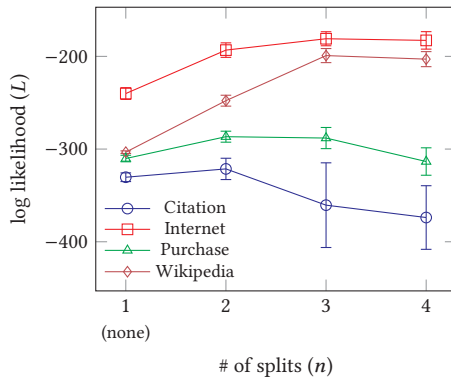

Figure 6: On real-world graphs, splitting nonterminal symbols ($n \geq 2$) always improves log-likelihood on the test graph, as compared to no splitting ($n = 1$), peaking at 2 or 3 splits and then dropping due to overfitting. Error bars indicate 95% confidence intervals. Higher is better.

show that this is indeed the case; the log-likelihood measure and the laHRG model pass the sanity check.

Next we extracted and tested the laHRG model on real world graphs. The log-likelihood results are illustrated in Fig. 6 for laHRG models of up to 4-splits. We find that the log-likelihood scores peak at $n = 2$ or 3 and then decreases when $n = 4$.

Recall that laHRG is the same as HRG [2] when $n = 1$. Based on the results from Fig. 6, we find that splitting does indeed increase HRG's ability to generate the test graph. However, as increasing n shows diminishing returns and sometimes decreases performance. The decrease in log likelihood when $n > 2$ is caused by model
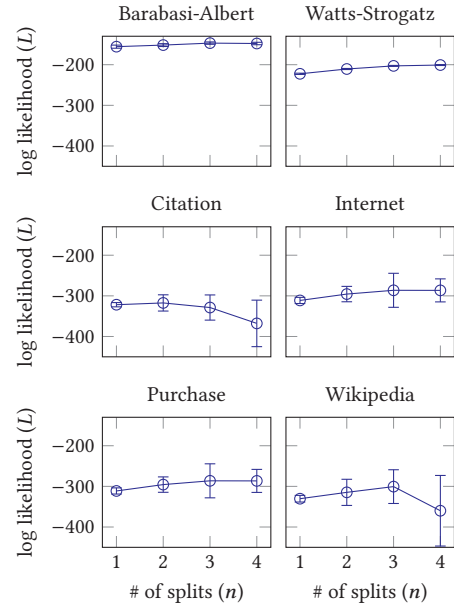
Figure 7: Loglikelihood on subgraphs that are disjoint from training graphs. Trends similar to Fig. 6 and Fig. 5 can be observed with this method. Error bars indicate 95% confidence intervals. Higher is better.

overfitting. The increase in node-splitting allows laHRG to fine-tune the rule probabilities to the training graph, which we find does not always generalize to the test graph.

5.2.2 Validate on Disjoint Subgraph. If it is difficult to find a test graph of the same type with the training graph, it is still possible to evaluate laHRG with the log likelihood metric. We can partition the graph into two disjoint parts, and use one for training and the other for testing. Fig. 7 shows the log likelihood of the test subgraphs that are disjoint from the training graphs. Again, splitting nonterminal symbols increase the log likelihood on the test graph, but as the number of splits (n) increases, log likelihood decreases due to overfitting.

5.3 Comparing against other Graph Generators

The log-likelihood metric is a principled approach to calculating the performance of a graph generator. Unfortunately, other graph generators are not capable of performing this type of likelihood calculation. In order to compare the laHRG graph model to other state of the art graph generators including the Kronecker [12] and Chung-Lu [8] models, we revert to traditional graph metrics to compare a generated graph to a test graph(not against the original graph).

Among the many choices for heuristic graph comparison metrics, we chose the degree distribution and graphlet correlation distance (GCD).

Recall that the sampling of 25-node subgraphs was necessary to ensure a non-zero probability for the log likelihood evaluation. No such requirement exists for evaluation on GCD. Nevertheless,

to maintain an apples to apples comparison, we performed similar graph sampling methods for degree distribution distance and GCD: we trained Kronecker and Chung-Lu models on a 25-node subgraph from the training graph, generated a 25-node graph, compared the generated graph against a 25-node subgraph of the test graph, and repeated this process 500 times.

As a baseline, we also compared the training and test graph directly to get a basic sense of their similarity. So, we directly compared 25-node subgraphs from the training graph to 25-node subgraphs of the test graph without any model. We repeated this direct comparison 500 times and report the mean and 95% confidence interval. We call this the "Direct" comparison because it does not involve any graph generation.

5.3.1 Degree Distribution Distance. In the present work we apply the degree distribution distance of Pržulj [18] to compare two or more degree distributions. Lower degree distribution distance between two graphs means they are more similar.

which is defined as follows. Given a graph H, we first scale and normalize the degree distribution of H:

$$S_H(k) = \frac{d_H(k)}{k}$$

$$T_H = \sum_{k=1}^{\infty} S_H(k)$$

$$N_H(k) = \frac{S_H(k)}{T_H}$$

in order to reduce the effect of higher degree nodes, where $d_H(K)$ is the number of nodes in H that have a degree of k. Then we calculate the distance between two degree distributions $D(d_H, d_{H'})$ as:

$$D(d_H, d_{H'}) = \frac{1}{\sqrt{2}} \sqrt{\sum_{k=1}^{\infty} (N_H(k) - N_{H'}(k))^2},$$

which is essentially a normalized sum of squares between the two distributions. We call this metric the degree distribution distance. Because this is a "distance" metric low values indicate high similarity.

Figure 8 illustrates the results of the degree distribution distance. Recall that the laHRG is identical to HRG [2] when $n = 1$. The Kronecker and Chung-Lu do not have an n parameter, so their plots are flat. All points represent the mean of 500 repetitions; each point contains error bars indicating the 95% confidence intervals – although many error bars are too small to be seen.

The laHRG model generates graphs that more closely follow the degree distribution of the test graph than graphs generated by Kronecker and Chung-Lu models. Higher nonterminal splitting, *i.e.*, $n > 1$, shows little change on the degree distribution distance.

It is expected that the Direct baseline outperforms all graph models, because the Direct baseline simply compares two graphs generated from the exact same generation process, which rewards an overfit model.

Here, the HRG models predict the test graph's degree distribution better than the Direct baseline does; whether this is because they generalize better, or due to chance, or some other reason, would need further analysis to determine. In any case, nonterminal splitting ($n \geq 2$) has only a slight effect on the model, generally
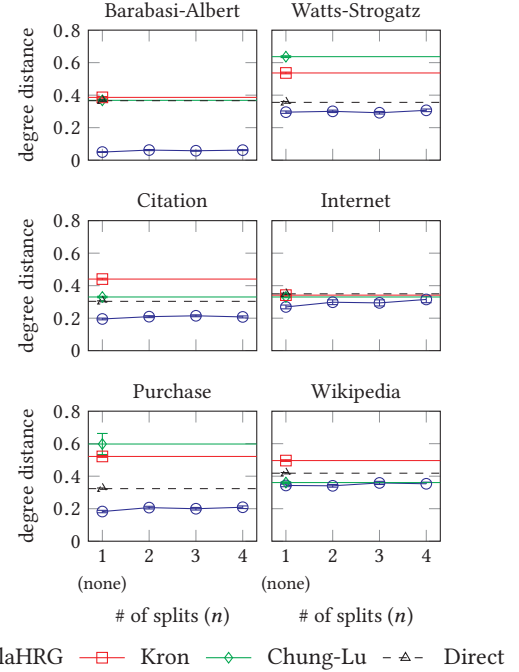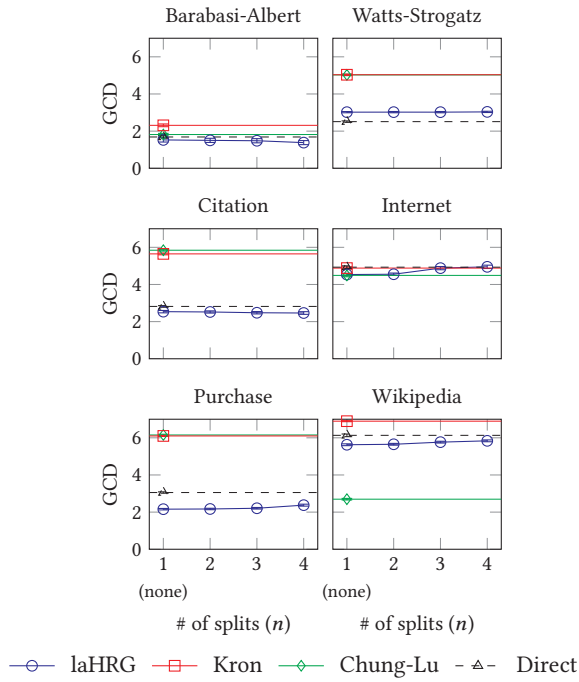

Figure 8: HRG models are shown to generate graphs with lower (= better) degree distribution distance to the test graph when compared to other models. Splitting nonterminals ($n \geq 2$) sometimes decreases degree distance but sometimes increases it.

attracting the degree distribution toward the training graph's and away from the test graph's.

Interestingly, the Direct baseline has similar or better performance than the Kronecker and Chung-Lu methods. It is unlikely that these results can be completely explained by overfitting. Instead, Kronecker or Chung-Lu methods may perform poorly due to underfitting, wherein these models do not model the training graph well enough. More work is needed to understand these results.

5.3.2 Graphlet Correlation Distance. Although the degree distribution is the most well known and widely adopted graph comparison metric, it is far from complete. The degree distribution can be easily mimicked by two very large and different networks. For example, previous work has shown that it is easy to construct two or more networks with exactly the same degree distribution but substantially different structure and function [14, 19]. There is mounting evidence which argues that the graphlet comparisons are a better way to measure the similarity between two graphs [18, 22]. Recent work from systems biology has identified a metric called the Graphlet Correlation Distance (GCD). The GCD computes the distance between two graphlet correlation matrices – one matrix for each graph [24]. Because the GCD is a distance metric, low values indicate high similarity where the GCD is 0 iff the two graphs are isomorphic.

Figure 9: HRG models are shown to generate graphs with lower (= better) graphlet correlation difference (GCD) to the test graph, when compared with other models. Splitting non-terminals ($n \geq 2$) sometimes inproves GCD and sometimes decreases it.

Figure 9 illustrates the results of the GCD. Recall that the laHRG is identical to HRG [10] when n = 1. The Kronecker and Chung-Lu do not have an n parameter, so their plots are flat. All points represent the mean of 500 repetitions; each point contains error bars indicating the 95 confidence intervals âĂŞ although many error bars are too small to be seen.

The Direct baseline illustrates how similar the training and test graphs are. As expected, we find that the Direct comparison is best on the random Watts-Strogatz graphs. But laHRG outperforms it on all of the real-world graphs.

5.4 Comparison with Log Likelihood Metric

GCD and Degree Distribution metrics indicate that laHRG is a better graph generator than other options like Kron and Chung-Lu graph generators, but our experiments seem to suggest that splitting nonterminals in HRG does not have much effect in terms of GCD and Degree Distribution. However, nonterminal splitting does increase log likelihood of the test graph, as explained in Section 5.2. This discrepancy is probably because log likelihood metric is able to capture more general structure and properties of a graph than GCD and Degree Distribution. Both GCD and Degree Distribution only focus on a specific graph property, which might not be perfectly correlated with overall structure of a graph. On the other hand, the log likelihood metric we propose does not overemphasize a

Table 2: Number of rules/parameters in grammars extracted from training graphs

Train	n			
	1	2	3	4
Citation	1,156	7,193	19,885	44,410
Internet	1,005	5,686	14,057	29,247
Purchase	969	6,196	18,237	38,186
Wikipedia	1,065	6,891	20,891	42,841
Barabasi-Albert	48	298	930	2,126
Watts-Strogatz	60	346	1,023	2,380

particular graph property, but directly measures the ability of the graph generator to generate the test graph.

6 GRAMMAR ANALYSIS

Recall that the HRG models merges two production rules if they are identical. Splitting rules produces subrules that have the same structure, but different symbols, so they cannot be merged; splitting nonterminal nodes will therefore increase the size of the grammar. In the worst case, the blowup could be cubic in n. Table 2 shows the sizes of all the grammars used in the experiments. Because rules with probability zero are excluded, the blowup is slightly less than cubic.

Here we see a trade-off between model size and performance. The larger the grammar gets, the better it is able to fit the training graph. On the other hand, we prefer smaller models to mitigate the possibility of overfitting. If we had not used separate training and test graphs, it would not be clear how to manage this trade-off, but our evaluation is able to demonstrate that larger grammars (up to a point) are indeed able to generalize to new data.

What do the HRG grammars look like? The models learned by unsupervised methods like EM can often be difficult to interpret, especially when the number of splits is high and the grammar is large. Figure 10 shows selected 2-split rules extracted from the as-topo training graph, namely, those with probability at least 0.1 and with at most two external nodes.

We can see that the subsymbols behave quite differently from each other. For example, the N_1^2 rule adds a connection between its two external nodes (via a third node), whereas none of the N_2^2 rules adds a connection (perhaps because, as can be seen in the RHS of the N^0 rule, they are already neighbors).

What can we learn from these graph grammars? This is an open question. If we assume that the tree decomposition provides a meaningful representation of the original graph, then we may be able to interrogate and assign meaning to these rules depending on their context. But we save this as a matter for future work.

7 CONCLUSION

This present work identifies and addresses two problems in applying Hyperedge Replacement Grammars (HRGs) to network data [2] by adding latent variables in order to make production rules more sensitive to context and by introducing a principled evaluation
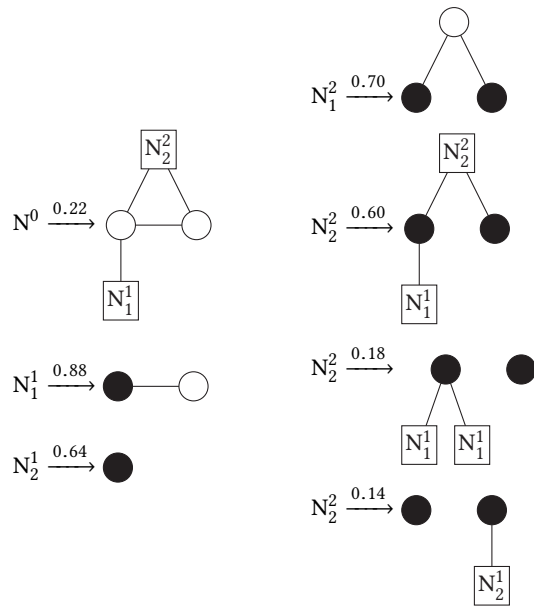
Figure 10: Rules extracted from as-topo, 2-split nontermi-nals, showing only those with probability at least 0.1 **and with at most two external nodes.**

methodology that computes the log likelihood that an HRG model generates a graph.

To guard against the possibility of the new model overfitting the original graph, we enforced a separation between the original graph from which the model is trained and a different graph on which the model is tested. This methodology should be better at selecting models that generalize well to new data. We confirmed Aguinaga et al.'s original finding that HRGs perform better than the widely-used Kronecker and Chung-Lu models, and showed that adding latent variables usually improves performance further.

Furthermore, we evaluated our method against the original HRG model by directly measuring the log-likelihood of the test graphs under all models. This metric is more principled than aggregation of statistics of select graph properties. Under this metric, our method improves over the original in all cases, peaking at either $n = 2$ or 3 splits.

HRGs extracted from tree decompositions are large. Splitting nonterminals grows the model even more. But our finding that 2- or 3-split grammars still generalize better to unseen graphs suggests that these models are not unreasonably large.

It remains for future work to test this claim by evaluating other generative graph models on test graphs distinct from training graphs. It should also be possible to simplify the HRG and laHRG models by trying to prune low-probability rules while maintaining high performance. Finally, more analysis is needed to provide an interpretation for the patterns automatically discovered by laHRGs.

REFERENCES

[1] Aaron B. Adcock, Blair D. Sullivan, and Michael W. Mahoney. 2016. Tree decompositions and social graphs. *Internet Mathematics* 12, 5 (2016), 315–361.

[2] Salvador Aguiñaga, Rodrigo Palacios, David Chiang, and Tim Weninger. 2016. Growing Graphs from Hyperedge Replacement Graph Grammars. In *Proc. CIKM.* 469–478.

[3] Salvador Aguinaga and Tim Weninger. 2016. The infinity mirror test for analyzing the robustness of graph generators. (2016). arXiv preprint arXiv:1606.04412.

[4] Nesreen K Ahmed, Jennifer Neville, Ryan A Rossi, and Nick Duffield. 2015. Efficient graphlet counting for large networks. In *Proc. ICDM.* 1–10.

[5] Deepayan Chakrabarti and Christos Faloutsos. 2006. Graph mining: Laws, generators, and algorithms. *Comput. Surveys* 38, 1 (2006), 2.

[6] David Chiang, Jacob Andreas, Daniel Bauer, Karl Moritz Hermann, Bevan Jones, and Kevin Knight. 2013. Parsing Graphs with Hyperedge Replacement Grammars.. In *Proc. ACL.* 924–932.

[7] Noam Chomsky. 1959. On certain formal properties of grammars. *Information and control* 2, 2 (1959), 137–167.

[8] Fan Chung and Linyuan Lu. 2002. Connected Components in Random Graphs with Given Expected Degree Sequences. *Annals of Combinatorics* 6, 2 (2002), 125âĂŞ145.

[9] A. P. Dempster, N. M. Laird, and D. B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B* 39, 1 (1977), 1–38.

[10] Vibhav Gogate and Rina Dechter. 2004. A Complete Anytime Algorithm for Treewidth. In *Uncertainty in Artificial Intelligence (UAI).* 201–208.

[11] K. Lari and S. J. Young. 1990. The estimation of stochastic context-free grammars using the Inside-Outside algorithm. *Computer Speech and Language* 4 (1990), 35–56.

[12] Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. 2010. Kronecker graphs: An approach to modeling networks. *Journal of Machine Learning Research* 11 (2010), 985–1042.

[13] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2005. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proc. KDD.* 177–187.

[14] Lun Li, David Alderson, John C Doyle, and Walter Willinger. 2005. Towards a theory of scale-free graphs: Definition, properties, and implications. *Internet Mathematics* 2, 4 (2005), 431–523.

[15] Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2005. Probabilistic CFG with Latent Annotations. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL '05).* Ann Arbor, Michigan, 75–82.

[16] Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. 2002. Network motifs: simple building blocks of complex networks. *Science* 298, 5594 (2002), 824–827.

[17] Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning Accurate, Compact, and Interpretable Tree Annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the ACL.* 433âĂŞ440.

[18] Nataša Pržulj. 2007. Biological network comparison using graphlet degree distribution. *Bioinformatics* 23, 2 (2007), e177–e183.

[19] Natasa Pržulj, Derek G Corneil, and Igor Jurisica. 2004. Modeling interactome: scale-free or geometric? *Bioinformatics* 20, 18 (2004), 3508–3515.

[20] Garry Robins, Pip Pattison, Yuval Kalish, and Dean Lusher. 2007. An introduction to exponential random graph (p*) models for social networks. *Social Networks* 29, 2 (2007), 173âĂŞ191.

[21] Comandur Seshadhri, Tamara G Kolda, and Ali Pinar. 2012. Community structure and scale-free collections of Erdős-Rényi graphs. *Physical Review E* 85, 5 (2012), 056109.

[22] Johan Ugander, Lars Backstrom, and Jon Kleinberg. 2013. Subgraph frequencies: Mapping the empirical and extremal geography of large graph collections. In *Proc. WWW.* 1307–1318.

[23] Ömer Nebil Yaveroğlu, Noël Malod-Dognin, Darren Davis, Zoran Levnajic, Vuk Janjic, Rasa Karapandza, Aleksandar Stojmirovic, and Nataša Pržulj. 2014. Revealing the hidden language of complex networks. *Scientific Reports* 4 (2014), 4547.

[24] Ömer Nebil Yaveroğlu, Tijana Milenković, and Nataša Pržulj. 2015. Proper evaluation of alignment-free network comparison methods. *Bioinformatics* 31, 16 (2015), 2697–2704.